MAiNGO - implemented functions

Jaromił Najman, Dominik Bongartz, Susanne Sass, Hatim Djelassi, Daniel Jungen, Alexander Mitsos*

Process Systems Engineering (AVT.SVT)
RWTH Aachen University
Forckenbeckstraße 51
52074 Aachen, Germany

Abstract: This short report summarizes non-elementary intrinsic functions currently implemented in the AVT.SVT in-house solver MAiNGO, where elementary instrinsic denotes functions such as addition, multiplication, division, exp, x^n. The latter can be found in the documentation of MC++. Using the non-elementary intrinsic functions (as opposed to implementing them by hand in the model using elementary functions) will in most cases result in tighter relaxations. In most cases, the `ALE`-syntax is also supported in `C++`.

1 Simple functions

1.1 square(x)

Form:	x^2
How to call in `C++`:	`sqr(x)`
How to call in `ALE`:	`sqr(x)`
Domain:	$\mathbb{R}$

1.2 $x \cdot \log(x)$

Form:	$x \cdot \log(x)$
How to call in `C++`:	`xlog(x)`, `xlogx(x)`
How to call in `ALE`:	`xlogx(x)`
Domain:	$x > 0$

1.3 $\exp(x) \cdot y$

Form:	$\exp(x) \cdot y$
How to call in `C++`:	`expx_times_y(x,y)`, `xexpy(y,x)`
How to call in `ALE`:	`xexpy(y,x)`
Domain:	$\mathbb{R}^2$
Remarks:	The formula can be found in [1]. Note that in `ALE`-syntax, the ordering of the variables is swapped.

Corresponding author: *A. Mitsos
Process Systems Engineering (AVT.SVT), RWTH Aachen University, Aachen, Germany
E-mail: amitsos@alum.mit.edu

1.4 $\frac{a_1 \cdot x}{a_2 \cdot x + \sum b_i \cdot y_i}$

Form:	$\frac{a_1 \cdot x}{a_2 \cdot x + \sum b_i \cdot y_i}$
How to call in `C++`:	`sum_div(vars, coeff)` where `vars` is a vector of size n and `coeff` is a vector of size $n+1$. It is `coeff[0]`$= a_1$, `coeff[1]`$= a_2$, `coeff[2]`$= b_1$, `coeff[3]`$= b_2$, ...
How to call in `ALE`:	`sum_div`$(x, y_1, \dots, y_n, a_1, a_2, b_1, \dots, b_n)$
Domain:	$(x, \mathbf{y}) =$ `vars` $\in \mathbb{R}^n_{>0}$, $a_i, b_i \in \mathbb{R}_{>0}$
Remarks:	The idea and derivation of the formula for 2D can be found in Section 2 of [9] or in [8].

1.5 $x \cdot \log(a \cdot x + \sum b_i \cdot y_i)$

Form:	$x \cdot \log(a \cdot x + \sum b_i \cdot y_i)$
How to call in `C++`:	`xlog_sum(vars, coeff)` where `vars` is a vector of size n and `coeff` is a vector of size n. It is `coeff[0]`$= a$, `coeff[1]`$= b_1$, `coeff[2]`$= b_2$, ...
How to call in `ALE`:	`xlog_sum`$(x, y_1, \dots, y_n, a, b_1, \dots, b_n)$
Domain:	$(x, \mathbf{y}) =$ `vars` $\in \mathbb{R}^n_{>0}$, $a, b_i \in \mathbb{R}_{>0}$
Remarks:	The idea and derivation of the formula for 2D can be found in [3].

1.6 $x \cdot \exp(a \cdot x)$

Form:	$x \cdot \exp(a \cdot x)$
How to call in `C++`:	`xexpax(x,a)`
How to call in `ALE`:	`xexpax(x,a)`
Domain:	$\mathbb{R}$
Remarks:	a is a constant. More details can be found in [4].

1.7 $\sqrt{x^2 + y^2}$

Form:	$\sqrt{x^2 + y^2}$
How to call in `C++`:	`euclidean_norm_2d(x,y)`, `norm2(x,y)`
How to call in `ALE`:	`norm2(x,y)`
Domain:	$\mathbb{R}^2$
Remarks:	This function is convex and not differentiable at $x = y = 0$.

1.8 Error function

Form:	$\frac{1}{\sqrt{\pi}} \int_{-x}^{x} \exp(-t^2) dt$
How to call in `C++`:	`erf(x)`
How to call in `ALE`:	`erf(x)`
Domain:	$\mathbb{R}$

1.9 Complementary error function

Form:	$1 - \frac{1}{\sqrt{\pi}} \int_{-x}^{x} \exp(-t^2)dt$
How to call in `C++`:	`erfc(x)`
How to call in `ALE`:	`erfc(x)`
Domain:	$\mathbb{R}$

1.10 $|x| \cdot x$

Form:	$\lvert x \rvert \cdot x$
How to call in `C++`:	`fabsx_times_x(x)`, `xabsx(x)`
How to call in `ALE`:	`xabsx(x)`
Domain:	$\mathbb{R}$

1.11 $\frac{x}{\sqrt{a+b\cdot x^2}}$

Form:	$\frac{x}{\sqrt{a+b\cdot x^2}}$
How to call in `C++`:	`regnormal(x,a,b)`
How to call in `ALE`:	`regnormal(x,a,b)`
Domain:	$x \in \mathbb{R}, a, b \in \mathbb{R}_{>0}$

1.12 Printing output

Form:	`std::cout << "Object Type #" << number << ":" << x << std::endl`
How to call in `C++`:	`mc_print(x,number)`
How to call in `ALE`:	N/A
Domain:	$\mathbb{R}$
Remarks:	This can be used for debugging. To reduce output turn off pre-processing.

2 Bounding functions

These functions can be used to exploit valid bounds that are known to the user. The functions cut off the relaxations at a given value.
It is up to the user to make sure the bounds are actually valid! If they are not, the resulting relaxations may be wrong, and MAiNGO may or may not detect this and (possibly) throw an exception.

2.1 Positive

Form:	[only cuts off the convex relaxation at ϵ]
How to call in `C++`:	`pos(x)`
How to call in `ALE`:	`pos(x)`
Domain:	$x \geqslant \epsilon > 0$
Remarks:	Used to make sure that the convex relaxation stays positive; ϵ is the McCormick tolerance specified in the settings file. If the function is evaluated at some $\hat{x} < \epsilon$, an exception is thrown. It is especially useful in cases when interval extensions provide nonpositive values.

2.2 Negative

Form:	[only cuts off the concave relaxation at $-\epsilon$]
How to call in `C++`:	`neg(x)`
How to call in `ALE`:	`neg(x)`
Domain:	$x \leqslant -\epsilon < 0$
Remarks:	Used to make sure that the concave relaxation stays negative; ϵ is the McCormick tolerance specified in the settings file. If the function is evaluated at some $\hat{x} > -\epsilon$, an exception is thrown. It is especially useful in cases when interval extensions provide nonnegative values.

2.3 Lower bounding function

Form:	[only cuts off the convex relaxation at lb]
How to call in `C++`:	`lb_func(x,lb)`
How to call in `ALE`:	`lb_func(x,lb)`
Domain:	$x \geqslant lb \in \mathbb{R}$
Remarks:	Used to make sure that the convex relaxation stays above lb. If the function is evaluated at some $\hat{x} < lb$, an exception is thrown. It is especially useful in cases when interval extensions provide values strictly lower than lb.

2.4 Upper bounding function

Form:	[only cuts off the concave relaxation at ub]
How to call in `C++`:	`ub_func(x,ub)`
How to call in `ALE`:	`ub_func(x,ub)`
Domain:	$x \leqslant ub \in \mathbb{R}$
Remarks:	Used to make sure that the concave relaxation stays under ub. If the function is evaluated at some $\hat{x} > ub$, an exception is thrown. It is especially useful in cases when interval extensions provide values strictly lower than ub.

2.5 Bounding function

Form:	[only cuts off the convex relaxation at lb and the concave relaxation at ub]
How to call in `C++`:	`bounding_func(x,lb,ub)`
How to call in `ALE`:	`bounding_func(x,lb,ub)`
Domain:	$lb \leqslant x \leqslant ub$ with $lb, x, ub \in \mathbb{R}$
Remarks:	Used to make sure that the relaxations stay between lb and ub. If the function is evaluated at some $\hat{x} \notin [lb, ub]$, an exception is thrown. It is especially useful in cases when interval extensions provide values not within $[lb, ub]$.

2.6 Squash node

Form:	[only cuts off the convex relaxation at lb and the concave relaxation at ub]
How to call in `C++`:	`squash_node(x,lb,ub)`, `squash(x,lb,ub)`
How to call in `ALE`:	`squash(x,lb,ub)`
Domain:	$lb \leqslant x \leqslant ub$ with $lb, x, ub \in \mathbb{R}$
Remarks:	Used to make sure that the relaxations stay between lb and ub. In order to properly use this function, the user ***has to*** define squash inequalities $lb \leqslant x \leqslant ub$. This function's main purpose is to reduce the number of variables and equality constraints. It works especially well when removing linear equality constraints. The only difference between `squash_node(x,lb,ub)` and `bounding_func(x,lb,ub)` is that the former will not raise an exception, since the user has to provide valid squash inequalities.

3 Physically motivated functions

3.1 Arrhenius function

Form:	$\exp(-\frac{k}{x})$
How to call in `C++`:	`arh(x,k)`
How to call in `ALE`:	`arh(x,k)`
Domain:	$x > 0$

3.2 Logarithmic mean temperature difference

Form:	$\frac{\Delta T_1 - \Delta T_2}{\log(\Delta T_1) - \log(\Delta T_2)}$
How to call in `C++`:	`lmtd(`ΔT_1`,`ΔT_2` )`
How to call in `ALE`:	`lmtd(`ΔT_1`,`ΔT_2` )`
Domain:	$\Delta T_1, \Delta T_2 > 0$
Remarks:	Used for heat exchangers. More details can be found in [2, 5].

3.3 Reciprocal of logarithmic mean temperature difference

Form:	$\frac{\log(\Delta T_1) - \log(\Delta T_2)}{\Delta T_1 - \Delta T_2}$
How to call in `C++`:	`rlmtd(`ΔT_1`,`ΔT_2`)`
How to call in `ALE`:	`rlmtd(`ΔT_1`,`ΔT_2`)`
Domain:	$\Delta T_1, \Delta T_2 > 0$
Remarks:	Used for heat exchangers. More details can be found in [2].

3.4 Vapor pressure

There are four functions for vapor pressure.
All functions are assumed to be convex and increasing.
While this is true for all physically reasonable parameter sets [4], it should be checked whether it holds in any given case (e.g., by plotting the function on a sufficiently fine grid).

1. Extended Antoine

Form:	$\exp\left(p_1 + \frac{p_2}{T+p_3} + p_4 \cdot T + p_5 \cdot \log(T) + p_6 \cdot T^{p_7}\right)$
How to call in `C++`:	`vapor_pressure(T,1,`p_1`,`p_2`,`p_3`,`p_4`,`p_5`,`p_6`,`p_7`)`, `ext_antoine_psat(T,`p_1`,`p_2`,`p_3`,`p_4`,`p_5`,`p_6`,`p_7`)`, `ext_antoine_psat(T,p)`, with `p` a vector of length 7
How to call in `ALE`:	`ext_antoine_psat(T,`p_1`,`p_2`,`p_3`,`p_4`,`p_5`,`p_6`,`p_7`)`
Domain:	$T > 0$
Remarks:	p_1–p_7 are constant parameters.

2. Antoine

Form:	$10^{p_1 - \frac{p_2}{p_3+T}}$
How to call in C++:	`vapor_pressure(T,2,`p_1`,`p_2`,`p_3`)`, `ext_antoine_psat(T,`p_1`,`p_2`,`p_3`)`, `ext_antoine_psat(T,p)`, with `p` a vector of length 3
How to call in ALE:	`antoine_psat(T,`p_1`,`p_2`,`p_3`)`
Domain:	$T > 0$
Remarks:	p_1–p_3 are constant parameters.

3. Wagner

Form:	$p_6 \cdot \exp\left(\frac{p_1\cdot(1-\frac{T}{T_c})+p_2\cdot(1-\frac{T}{T_c})^{1.5}+p_3\cdot(1-\frac{T}{T_c})^3+p_4\cdot(1-\frac{T}{T_c})^6}{\frac{T}{T_c}}\right)$
How to call in C++:	`vapor_pressure(T,3,`p_1`,`p_2`,`p_3`,`p_4`,`T_c`,`p_6`)`, `wagner_psat(T,`p_1`,`p_2`,`p_3`,`p_4`,`T_c`,`p_6`)`, `wagner_psat(T,p)`, with `p` a vector of length 6
How to call in ALE:	`wagner_psat(T,`p_1`,`p_2`,`p_3`,`p_4`,`T_c`,`p_6`)`
Domain:	$T > 0$
Remarks:	p_1–p_4 and T_c are constant parameters.

4. IK-CAPE

Form:	$\exp(p_1 + p_2\cdot T + p_3\cdot T^2 + p_4\cdot T^3 + p_5\cdot T^4 + p_6\cdot T^5 + p_7\cdot T^6 + p_8\cdot T^7 + p_9\cdot T^8 + p_{10}\cdot T^9)$
How to call in C++:	`vapor_pressure(T,4,`p_1`,`p_2`,`p_3`,`p_4`,`p_5`,`p_6`,`p_7`,`p_8`,`p_9`,`p_{10}`)`, `ik_cape_psat(T,`p_1`,`p_2`,`p_3`,`p_4`,`p_5`,`p_6`,`p_7`,`p_8`,`p_9`,`p_{10}`)`, `ik_cape_psat(T,p)`, with `p` a vector of length 10
How to call in ALE:	`ik_cape_psat(T,`p_1`,`p_2`,`p_3`,`p_4`,`p_5`,`p_6`,`p_7`,`p_8`,`p_9`,`p_{10}`)`
Domain:	$T > 0$
Remarks:	p_1–p_{10} are constant parameters.

3.5 Saturation temperature

There are four functions for saturation temperature, which are the inverse of the vapor pressure functions listed above. Currently only the Antoine saturation temperature is implemented.

All functions are assumed to be concave and increasing.

While this is true for all physically reasonable parameter sets [4], it should be checked whether it holds in any given case (e.g., by plotting the function on a sufficiently fine grid).

1. Extended Antoine

Form:	$\exp\left(p_1 + \frac{p_2}{T+p_3} + p_4 \cdot T + p_5 \cdot \log(T) + p_6 \cdot T^{p_7}\right)^{-1}$
How to call in `C++`:	Not implemented
How to call in `ALE`:	Not implemented
Domain:	$T > 0$
Remarks:	p_1–p_7 are constant parameters. The computation of the inverse is performed numerically.

2. Antoine

Form:	$\frac{p_2}{p_1 - \frac{\log(T)}{\log(10)}} - p_3$
How to call in `C++`:	`saturation_temperature(T,2,`p_1`,`p_2`,`p_3`)`, `antoine_tsat(T,`p_1`,`p_2`,`p_3`)`, `antoine_tsat(T,p)`, with `p` a vector of length 3
How to call in `ALE`:	`antoine_tsat(T,`p_1`,`p_2`,`p_3`)`
Domain:	$T > 0$
Remarks:	p_1–p_3 are constant parameters.

3. Wagner

Form:	$\exp\left(\frac{p_1 \cdot (1-\frac{T}{T_c}) + p_2 \cdot (1-\frac{T}{T_c})^{1.5} + p_3 \cdot (1-\frac{T}{T_c})^3 + p_4 \cdot (1-\frac{T}{T_c})^6}{\frac{T}{T_c}}\right)^{-1}$
How to call in `C++`:	Not implemented
How to call in `ALE`:	Not implemented
Domain:	$T > 0$
Remarks:	p_1–p_4 and T_c are constant parameters. The computation of the inverse is performed numerically.

4. IK-CAPE

Form:	$\exp(p_1 + p_2 \cdot T + p_3 \cdot T^2 + p_4 \cdot T^3 + p_5 \cdot T^4 + p_6 \cdot T^5 + p_7 \cdot T^6 + p_8 \cdot T^7 + p_9 \cdot T^8 + p_{10} \cdot T^9)^{-1}$
How to call in `C++`:	Not implemented
How to call in `ALE`:	Not implemented
Domain:	$T > 0$
Remarks:	p_1–p_{10} are constant parameters. The computation of the inverse is performed numerically.

3.6 Ideal gas enthalpy

There are four functions for ideal gas enthalpy, which correspond to an integration over four different heat capacity models from T_0 to T.
All functions are assumed to be convex and increasing.
While for all physically reasonable parameter sets it should be increasing and in most cases it should also be convex [4], it should be checked whether it holds in any given case (e.g., by plotting the function on a sufficiently fine grid).

1. Aspen polynomial

Form:	$\int_{T_0}^{T} p_1 + p_2 \cdot \tilde{T} + p_3 \cdot \tilde{T}^2 + p_4 \cdot \tilde{T}^3 + p_5 \cdot \tilde{T}^4 + p_6 \cdot \tilde{T}^5 d\tilde{T}$
How to call in C++:	ideal_gas_enthalpy(T,T_0,1,p_1,p_2,p_3,p_4,p_5,p_6), aspen_hig(T,T_0,p_1,p_2,p_3,p_4,p_5,p_6), aspen_hig(T,T_0,p), with p a vector of length 6
How to call in ALE:	aspen_hig(T,T_0,p_1,p_2,p_3,p_4,p_5,p_6)
Domain:	$T > 0$
Remarks:	p_1–p_6 are constant parameters, T_0 is the reference temperature.

2. NASA 9-Coefficient

Form:	$\int_{T_0}^{T} \frac{p_1}{\tilde{T}^2} + \frac{p_2}{\tilde{T}} + p_3 + p_4 \cdot \tilde{T} + p_5 \cdot \tilde{T}^2 + p_6 \cdot \tilde{T}^3 + p_7 \cdot \tilde{T}^4 d\tilde{T}$
How to call in C++:	ideal_gas_enthalpy(T,T_0,2,p_1,p_2,p_3,p_4,p_5,p_6,p_7), nasa9_hig(T,T_0,p_1,p_2,p_3,p_4,p_5,p_6,p_7), nasa9_hig(T,T_0,p), with p a vector of length 7
How to call in ALE:	nasa9_hig(T,T_0,p_1,p_2,p_3,p_4,p_5,p_6,p_7)
Domain:	$T > 0$
Remarks:	p_1–p_7 are constant parameters, T_0 is the reference temperature.

3. DIPPR 107

Form:	$\int_{T_0}^{T} p_1 + p_2 \cdot \left(\frac{\frac{p_3}{\tilde{T}}}{\sinh(\frac{p_3}{\tilde{T}})} \right)^2 + p_4 \cdot \left(\frac{\frac{p_5}{\tilde{T}}}{\cosh(\frac{p_5}{\tilde{T}})} \right)^2 d\tilde{T}$
How to call in C++:	ideal_gas_enthalpy(T,T_0,3,p_1,p_2,p_3,p_4,p_5), dippr107_hig(T,T_0,p_1,p_2,p_3,p_4,p_5), dippr107_hig(T,T_0,p), with p a vector of length 5
How to call in ALE:	dippr107_hig(T,T_0,p_1,p_2,p_3,p_4,p_5)
Domain:	$T > 0$
Remarks:	p_1–p_5 are constant parameters, T_0 is the reference temperature.

4. DIPPR 127

Form:	$\int_{T_0}^{T} p_1 + p_2 \cdot \left(\left(\frac{p_3}{\tilde{T}}\right)^2 \cdot \frac{\exp(\frac{p_3}{\tilde{T}})}{\exp(\frac{p_3}{\tilde{T}}-1)^2} \right) + p_4 \cdot \left(\left(\frac{p_5}{\tilde{T}}\right)^2 \cdot \frac{\exp(\frac{p_5}{\tilde{T}})}{\exp(\frac{p_5}{\tilde{T}}-1)^2} \right) + p_6 \cdot \left(\left(\frac{p_7}{\tilde{T}}\right)^2 \cdot \frac{\exp(\frac{p_7}{\tilde{T}})}{\exp(\frac{p_7}{\tilde{T}}-1)^2} \right) d\tilde{T}$
How to call in C++:	ideal_gas_enthalpy(T,T_0,4,p_1,p_2,p_3,p_4,p_5,p_6,p_7), dippr127_hig(T,T_0,p_1,p_2,p_3,p_4,p_5,p_6,p_7), dippr127_hig(T,T_0,p), with p a vector of length 7
How to call in ALE:	dippr127_hig(T,T_0,p_1,p_2,p_3,p_4,p_5,p_6,p_7)
Domain:	$T > 0$
Remarks:	p_1–p_7 are constant parameters, T_0 is the reference temperature.

3.7 Enthalpy of vaporization

There are two functions for enthalpy of vaporization. The functional forms listed below are only used for $T \leqslant T_c$. For $T > T_c$, the enthalpy of vaporization is set to zero. This makes it possible to extrapolate correctly beyond the critical point.
All functions are assumed to be concave and decreasing below T_c.
While for all physically reasonable parameter sets it should be decreasing and in most cases it should also be concave [4], it should be checked whether it holds in any given case (e.g., by plotting the function on a sufficiently fine grid).

1. Watson

Form:	$dHT_1 \cdot (\frac{1-\frac{T}{T_c}}{1-\frac{T_1}{T_c}})^{a+b\cdot(1-\frac{T}{T_c})}$
How to call in `C++`:	`enthalpy_of_vaporization(T,1,`T_c`,`a`,`b`,`T_1`,`dHT_1`)`,
	`watson_dhvap(T,`T_c`,`a`,`b`,`T_1`,`dHT_1`)`
	`watson_dhvap(T,p)`, with `p` a vector of length 5
How to call in `ALE`:	`watson_dhvap(T,`T_c`,`a`,`b`,`T_1`,`dHT_1`)`
Domain:	$T > 0$
Remarks:	T_c, a, b, T_1, dHT_1 are constant parameters.

2. DIPPR

Form:	$p_1 \cdot (1 - \frac{T}{T_c})^{p_2+p_3\cdot(\frac{T}{T_c})^2+p_5\cdot(\frac{T}{T_c})^3}$
How to call in `C++`:	`enthalpy_of_vaporization(T,2,`T_c`,`p_1`,`p_2`,`p_3`,`p_4`,`p_5`)`,
	`dippr106_dhvap(T,`T_c`,`p_1`,`p_2`,`p_3`,`p_4`,`p_5`)`,
	`dippr106_dhvap(T,p)`, with `p` a vector of length 6
How to call in `ALE`:	`dippr106_dhvap(T,`T_c`,`p_1`,`p_2`,`p_3`,`p_4`,`p_5`)`
Domain:	$T > 0$
Remarks:	p_1–p_5 and T_c are constant parameters.

3.8 Functions for NRTL

All analyzes can be found in [4].

3.8.1 τ

Form:	$a + \frac{b}{T} + e \cdot \log T + f \cdot T$
How to call in C++:	`nrtl_tau(T,a,b,e,f)`, `nrtl_tau(T,p)`, with `p` a vector of length 4
How to call in ALE:	`nrtl_tau(T,a,b,e,f)`
Domain:	$T > 0$
Remarks:	a, b, e, f are constant parameters.

3.8.2 $\frac{d\tau}{dT}$

Form:	$-\frac{b}{T^2} + \frac{e}{T} + f$
How to call in C++:	`nrtl_dtau(T,b,e,f)`, `nrtl_dtau(T,p)`, with `p` a vector of length 3
How to call in ALE:	`nrtl_dtau(T,b,e,f)`
Domain:	$T > 0$
Remarks:	b, e, f are constant parameters.

3.8.3 G

Form:	$\exp(-\alpha \cdot (a + \frac{b}{T} + e \cdot \log T + f \cdot T))$
How to call in C++:	`nrtl_G(T,a,b,e,f,`α`)`, `nrtl_g(T,a,b,e,f,`α`)`, `nrtl_g(T,p)`, with `p` a vector of length 5
How to call in ALE:	`nrtl_g(T,a,b,e,f,`α`)`
Domain:	$T > 0$
Remarks:	a, b, e, f, α are constant parameters.

3.8.4 $G \cdot \tau$

Form:	$\exp(-\alpha \cdot (a + \frac{b}{T} + e \cdot \log T + f \cdot T)) \cdot (a + \frac{b}{T} + e \cdot \log T + f \cdot T)$
How to call in C++:	`nrtl_Gtau(T,a,b,e,f,`α`)`, `nrtl_gtau(T,a,b,e,f,`α`)`, `nrtl_gtau(T,p)`, with `p` a vector of length 5
How to call in ALE:	`nrtl_gtau(T,a,b,e,f,`α`)`
Domain:	$T > 0$
Remarks:	a, b, e, f, α are constant parameters.

3.8.5 $G \cdot \frac{d\tau}{dT}$

Form:	$\exp(-\alpha \cdot (a + \frac{b}{T} + e \cdot \log T + f \cdot T)) \cdot (-\frac{b}{T^2} + \frac{e}{T} + f)$
How to call in C++:	`nrtl_Gdtau(T,a,b,e,f,`α`)`, `nrtl_gdtau(T,a,b,e,f,`α`)`, `nrtl_gdtau(T,p)`, with `p` a vector of length 5
How to call in ALE:	`nrtl_gdtau(T,a,b,e,f,`α`)`
Domain:	$T > 0$
Remarks:	a, b, e, f, α are constant parameters. Monotonicity and convexity are determined heuristically for $e, f \neq 0$.

3.8.6 $\frac{dG}{dT} \cdot \tau$

Form:	$-\alpha \cdot \exp(-\alpha \cdot (a + \frac{b}{T} + e \cdot \log T + f \cdot T)) \cdot (-\frac{b}{T^2} + \frac{e}{T} + f) \cdot (a + \frac{b}{T} + e \cdot \log T + f \cdot T)$
How to call in `C++`:	`nrtl_dGtau(T,a,b,e,f,`α`)`, `nrtl_dgtau(T,a,b,e,f,`α`)`, `nrtl_dgtau(T,p)`, with `p` a vector of length 5
How to call in `ALE`:	`nrtl_dgtau(T,a,b,e,f,`α`)`
Domain:	$T > 0$
Remarks:	a, b, e, f, α are constant parameters. Monotonicity and convexity are determined heuristically.

3.9 Schroeder functions

The following functions implement special correlations for thermodynamic properties of ethanol [6].

Functions 3.9.1 and 3.9.2 are convex and increasing for $T \leqslant 514.71K$. Function 3.9.3 is concave and decreasing for $290.3 \leqslant T \leqslant 514.71K$.

3.9.1 p saturation ethanol

Form:	$62.68 \cdot \exp(\frac{514.71}{T} \cdot (-8.94161 \cdot (1 - \frac{T}{514.71}) + 1.61761 \cdot (1 - \frac{T}{514.71})^{1.5} - 51.1428 \cdot (1 - \frac{T}{514.71})^{3.4} + 53.1360 \cdot (1 - \frac{T}{514.71})^{3.7}))$
How to call in C++:	`p_sat_ethanol_schroeder(T)`, `schroeder_ethanol_p(T)`
How to call in ALE:	`schroeder_ethanol_p(T)`
Domain:	$T > 514.71$

3.9.2 ρ vapor saturation ethanol

Form:	$273.195 \cdot \exp(-1.75362 \cdot (1 - \frac{T}{514.71})^{0.21} - 10.5323 \cdot (1 - \frac{T}{514.71})^{1.1} - 37.6407 \cdot (1 - \frac{T}{514.71})^{3.4} - 129.762 \cdot (1 - \frac{T}{514.71})^{10}))$
How to call in C++:	`rho_vap_sat_ethanol_schroeder(T)`, `schroeder_ethanol_rhovap(T)`
How to call in ALE:	`schroeder_ethanol_rhovap(T)`
Domain:	$T > 514.71$

3.9.3 ρ liquid saturation ethanol

Form:	$273.195 \cdot \exp(9.00921 \cdot (1 - \frac{T}{514.71})^{0.5} - 23.1668 \cdot (1 - \frac{T}{514.71})^{0.8} + 30.9092 \cdot (1 - \frac{T}{514.71})^{1.1} - 16.5459 \cdot (1 - \frac{T}{514.71})^{1.5} + 3.64294 \cdot (1 - \frac{T}{514.71})^{3.3}))$
How to call in C++:	`rho_liq_sat_ethanol_schroeder(T)`, `schroeder_ethanol_rholiq(T)`
How to call in ALE:	`schroeder_ethanol_rholiq(T)`
Domain:	$T > 514.71$

4 Cost functions

Currently, only one cost function is implemented.

1. Guthrie

Form:	$10^{p_1+p_2\cdot\log_{10}(x)+p_3\cdot(\log_{10}(x))^2}$
How to call in `C++`:	`cost_function(x,1,`p_1`,`p_2`,`p_3`)`, `cost_turton(x,`p_1`,`p_2`,`p_3`)`, `cost_turton(x,p)`, with `p` a vector of length 3
How to call in `ALE`:	`cost_turton(x,`p_1`,`p_2`,`p_3`)`
Domain:	$x > 0$
Remarks:	Function for equipment costing based on a characteristic sizing variable x. p_1–p_3 are constant parameters. The analysis of this function can be found in [4]. More information on the Guthrie cost function can be found in [10].

5 Neural Networks

5.1 Hyperbolic tangent

Form:	$\tanh(x)$
How to call in `C++`:	`tanh(x)`
How to call in `ALE`:	`tanh(x)`
Domain:	$\mathbb{R}$
Remarks:	More information can be found in [7].

6 Gaussian Processes

6.1 Covariance functions

1. Matérn $\frac{1}{2}$

Form:	$\exp(-\sqrt{x})$
How to call in `C++`:	`covariance_function(x,1)`, `covar_matern_1(x)`
How to call in `ALE`:	`covar_matern_1(x)`
Domain:	$x \geqslant 0$

2. Matérn $\frac{3}{2}$

Form:	$(1+\sqrt{3}\sqrt{x})\cdot\exp(-\sqrt{3}\sqrt{x})$
How to call in `C++`:	`covariance_function(x,2)`, `covar_matern_3(x)`
How to call in `ALE`:	`covar_matern_3(x)`
Domain:	$x \geqslant 0$

3. Matérn $\frac{5}{2}$

Form:	$(1+\sqrt{5}\sqrt{x}+\frac{5}{3}x)\cdot\exp(-\sqrt{5}\sqrt{x})$
How to call in `C++`:	`covariance_function(x,3)`, `covar_matern_5(x)`
How to call in `ALE`:	`covar_matern_5(x)`
Domain:	$x \geqslant 0$

4. Squared Exponential

Form:	$\exp(-\frac{1}{2}x)$
How to call in `C++`:	`covariance_function(x,4)`, `covar_sqrexp(x)`
How to call in `ALE`:	`covar_sqrexp(x)`
Domain:	$x \geqslant 0$

6.2 Acquisition Functions

1. Lower Confidence Bound

Form:	$\mu-\kappa\cdot\sigma$
How to call in `C++`:	`acquisition_function(`μ`,`σ`,1,`κ`)`, `af_lcb(`μ`,`σ`,`κ`)`
How to call in `ALE`:	`af_lcb(`μ`,`σ`,`κ`)`
Domain:	$\mu\in\mathbb{R}, \sigma \geqslant 0$

2. Expected Improvement

Form:

$$\begin{cases} (f_{\min}-\mu)\cdot\left(\dfrac{\operatorname{erf}\left(\frac{\frac{f_{\min}-\mu}{\sigma}}{\sqrt{2}}\right)}{2}+\frac{1}{2}\right)+\sigma\cdot\dfrac{\exp\left(-\frac{\left(\frac{f_{\min}-\mu}{\sigma}\right)^2}{2}\right)}{\sqrt{2\cdot\pi}} & ,\sigma>0 \\ \max\{f_{\min}-\mu,0\} & ,\sigma=0 \end{cases}$$

How to call in `C++`:	`acquisition_function(`μ`,`σ`,2,`$f_{\min}$`)`, `af_ei(`μ`,`σ`,`$f_{\min}$`)`
How to call in `ALE`:	`af_ei(`μ`,`σ`,`$f_{\min}$`)`
Domain:	$\mu\in\mathbb{R}, \sigma \geqslant 0$

3. Probability of Improvement

Form:	$\begin{cases} \frac{\operatorname{erf}\left(\frac{\frac{f_{\min}-\mu}{\sigma}}{\sqrt{2}}\right)}{2}+\frac{1}{2} & ,\sigma>0 \\ 0 & ,\sigma=0,\ f_{\min}\leqslant\mu \\ 1 & ,\sigma=0,\ f_{\min}>\mu \end{cases}$
How to call in `C++`:	`acquisition_function(`μ`,`σ`,3,`$f_{\min}$`)`, `af_pi(`μ`,`σ`,`$f_{\min}$`)`
How to call in `ALE`:	`af_pi(`μ`,`σ`,`$f_{\min}$`)`
Domain:	$\mu\in\mathbb{R},\sigma\geqslant 0$

6.3 Probability Functions

Gaussian Probability Density Function

Form:	$\frac{1}{\sqrt{2\pi}}\cdot\exp(-\frac{x^2}{2})$
How to call in `C++`:	`gaussian_probability_density_function(x)`, `gpdf(x)`
How to call in `ALE`:	`gpdf(x)`
Domain:	$x\in\mathbb{R}$

References

[1] A. Khajavirad and N. V. Sahinidis. Convex envelopes generated from finitely many compact convex sets. *Mathematical Programming*, 137(1-2):371–408, 2013.

[2] M. Mistry and R. Misener. Optimising heat exchanger network synthesis using convexity properties of the logarithmic mean temperature difference. *Computers & Chemical Engineering*, 94:1 – 17, 2016.

[3] J. Najman, D. Bongartz, and A. Mitsos. Convex relaxations of componentwise convex functions. *Computers & Chemical Engineering*, 130:106527, 2019.

[4] J. Najman, D. Bongartz, and A. Mitsos. Relaxations of thermodynamic property and costing models in process engineering. *Computers & Chemical Engineering*, 130:106571, 2019.

[5] J. Najman and A. Mitsos. Convergence order of McCormick relaxations of LMTD function in heat exchanger networks. In Z. Kravanja and M. Bogataj, editors, *26th European Symposium on Computer Aided Process Engineering*, volume 38 of *Computer Aided Chemical Engineering*, pages 1605 – 1610. Elsevier, 2016.

[6] J. Schroeder, S. Penoncello, and J. Schroeder. A fundamental equation of state for ethanol. *Journal of Physical and Chemical Reference Data*, 43(4):043102, 2014.

[7] A. M. Schweidtmann and A. Mitsos. Global Deterministic Optimization with Artificial Neural Networks Embedded. *Journal of Optimization Theory and Applications*, 180(3), 2019.

[8] M. Tawarmalani and N. V. Sahinidis. Semidefinite relaxations of fractional programs via novel convexification techniques. *Journal of Global Optimization*, 20(2):133–154, 2001.

[9] M. Tawarmalani and N. V. Sahinidis. *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications*, volume 65. Springer Science & Business Media, 2002.

[10] R. Turton, R. Bailie, W. Whiting, and J. Shaeiwitz. *Analysis, Synthesis, and Design of Chemical Processes*. Prentice Hall PTR, Upper Saddle River, NJ., 2009.